(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平9－330233

(43)公開日　平成９年(1997)12月22日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/45			Ｇ０６Ｆ　9/44	３２２Ｆ	
11/28	３３０		11/28	３３０Ａ	
			9/44	３２２Ｍ	

審査請求　未請求　請求項の数１　ＯＬ　（全 15 頁）

(21)出願番号　特願平9－55902

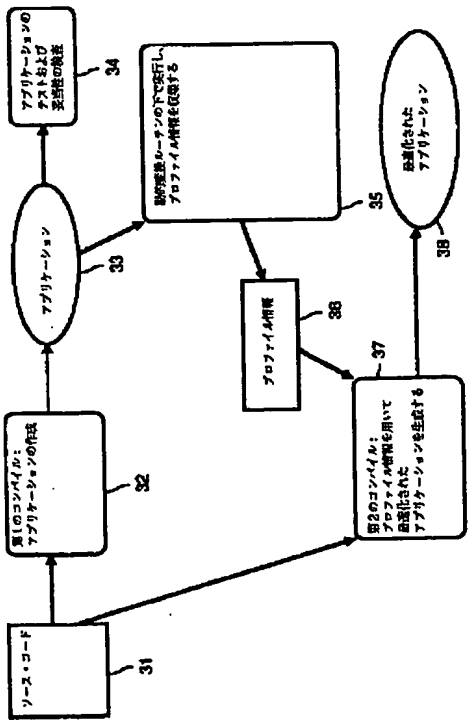
(22)出願日　平成９年(1997)３月11日

(31)優先権主張番号　６１６，６０８
(32)優先日　1996年３月15日
(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000400
ヒューレット・パッカード・カンパニー
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000
(72)発明者　ウィリアム・ピー・バズビー
アメリカ合衆国94019カリフォルニア州ハーフ・ムーン・ベイ、カサ・デル・マー・ドライブ　404
(74)代理人　弁理士　岡田　次生

(54)【発明の名称】　最適目的コード生成方法

(57)【要約】
【課題】プロファイルに基づいて目的コードを最適化する場合のコンパイル回数を削減し、コンパイルの効率化を図る。
【解決手段】アプリケーションのソース・コードをコンパイルして、第１の目的コードを生成する。アプリケーション最適化のために必要となるプロファイル情報を生成する機能を持つプロファイル作成コードを含むように、本発明の動的変換ルーチンによって第１の目的コードを第２の目的コードに動的に変換する。次に、第２の目的コードを実行して、上記プロファイル作成コードに基づいて、プロファイル情報を生成する。最後に、このプロファイル情報を使用してアプリケーションのソース・コードを再コンパイルし、最適化された目的コードを生成する。



1

【特許請求の範囲】

【請求項１】アプリケーションに関する最適化された目的コードを生成する方法であって、

アプリケーションのソース・コードをコンパイルして該アプリケーションに関する第１の目的コードを生成するステップと、

実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを含む第２の目的コードに上記第１の目的コードを変換し、該第２の目的コードを実行してプロファイル情報を生成することを含む上記第１の目的コードに関するプロファイル情報を生成するステップと、

上記プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成するステップと、

を含む最適目的コード生成方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、コンパイラ最適化に関するもので、特に最適化コンパイル・システムにおいて実行時情報を収集し活用するための動的変換の使用に関するものである。

【０００２】

【従来の技術】プログラムは、一般に高水準プログラム言語で書かれる。しばしばソース・コードと呼ばれるこの高水準言語は、コンパイラ・プログラムによってアセンブリ言語に変換される。目的コードと呼ばれるバイナリ形式のアセンブリ言語は、コンピュータによって実際に実行されるコード形式である。目的コードは、先ず、リンカによって相互にリンクされる目的コード・モジュールの形式で生成される。本明細書において、用語「コンパイル」は、目的コード・モジュールを生成するプロセスおよび目的コード・モジュールを相互にリンクするプロセスの両者を含む。

【０００３】目標コンピュータによって実行される目的コードの処理性能を向上させるため、コンパイルされたコードはしばしば最適化される。目的コードはいくつかの異なる形態で最適化される。

【０００４】最適化の１つのタイプは、プロファイルに基づく最適化(profile-based optimizationの頭文字をとってＰＢＯと呼ばれる)である。ＰＢＯにおいては、コンパイラは、典型的な入力データセットを用い、アプリケーションの実行を通して得られるプロファイル・データに基づいてアプリケーションを最適化する。例えば、特定の複数のプロシージャが頻繁に互いを呼び出すとすれば、リンカは、目的コード・ファイルの中でそれらプロシージャを互いに近い位置に置き、それにより、アプリケーション実行中の命令キャッシュ・ミス、変換ルックアサイド・バッファ(ＴＬＢ)ミス、および記憶ページ・フォールトを減少させることができる。同様の最

2

適化は、プロシージャの基本ブロックのレベルでも行うことができる。基本ブロックまたは分節は、開始点および終了点以外には分岐が存在しないようなアセンブリ・コードの１つの連続的節である。プロファイル・データは、また、コード・スケジューリングおよびレジスタ割り付けのような他の一般的タスクのためにもコンパイラによって使用される。

【０００５】ＰＢＯを実行するためには、アプリケーションのソース・コードは、プロファイル作成用の特別コードを挿入することによって目的コードに仕組みを組み込む特別なコンパイラ・オプションを使用してコンパイルされる。アプリケーションが実行されると、その特別プロファイル作成コードによってプロファイル・データが作成される。次にプロファイル・データが収集される。アプリケーションのソース・コードが再コンパイルされる時、プロファイル・データを使用して、アプリケーション・ソース・コードの新しいコンパイルが一層最適化される。ＰＢＯの詳細は、例えば、ヒューレット・パッカード社発行の"Programming on HP-UX, HP 9000 Series 700/800 Computers"(HP Part No. B2355-90652, January 1995, pp. 8-1 through 8-17)に記載されている。

【０００６】

【発明が解決しようとする課題】ＰＢＯの使用にかかわる１つの問題は、ＰＢＯのための別のコンパイルの実行が必要とされる点である。従って、ユーザがＰＢＯを利用することを望む場合、典型的にはアプリケーション・ソース・コードは３回コンパイルされることが必要である。第１回は、アプリケーションをテストし妥当性を検査するために使用されるアプリケーションのコピーを作成するためのものである。コードの第２回のコンパイルでは、プロファイル・データを作成するためプロファイル作成用特別コードが追加される。次に、コンパイルされたアプリケーションを最適化するため、生成されたプロファイル・データを使用してアプリケーションを構築する第３のコンパイルが必要である。このように独立したコンパイルを行うことなくＰＢＯを実施できる方法が求められている。

【０００７】

【課題を解決するための手段】本発明に従う動的変換を使用して、アプリケーションの目的コードを最適化するために使用されるプロファイル情報が生成される。アプリケーションに関する最適化された目的コードを生成するため、アプリケーションのソース・コードがコンパイルされ、アプリケーションに関する第１の目的コードが生成される。第１の目的コードは、アプリケーションをテストして妥当性を検査するために使用することもできる。第１の目的コードは、また、アプリケーションに関するプロファイル情報の生成に使用することもできる。これは、第１の目的コードを第２の目的コードに動的に

変換することによって実施される。この変換の際に、実行時にプロファイル情報を生成するプロファイル作成コードが第２の目的コードに、含められる。生成された第２の目的コードは次にプロファイル情報を生成するため実行される。最後に、アプリケーションのソース・コードは、最適化された目的コードを作成するため、生成したプロファイル情報を活用して再コンパイルされる。

【０００８】本発明は、具体的には、アプリケーションに関する最適化された目的コードを生成するため、アプリケーションのソース・コードをコンパイルして該アプリケーションに関する第１の目的コードを生成するステップ、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを含む第２の目的コードに上記第１の目的コードを変換し、該第２の目的コードを実行してプロファイル情報を生成することを含む上記第１の目的コードに関するプロファイル情報を生成するステップ、および上記プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成するステップを含む最適化された目的コード生成方法を提供する。

【０００９】アプリケーションの目的コードを更に最適化するため最適化された目的コードに関する追加のプロファイル情報を生成するため、このプロセスを繰り返すこともできる。

【００１０】更に、本発明の別の局面では、上記第１の目的コードに注釈が配置される。コンパイラの変換ルーチンが第１の目的コード内の注釈を利用して、第２の目的コード内に置かれるべき特定のプロファイル作成コードに従って生成されるプロファイル情報を決定する。注釈は、コンパイラによって第１の目的コードに置かれる。代替的にあるいは追加として、目的コード内の注釈および目的コードの実行の結果生成されるプロファイル情報に対するアクセス機能をユーザに与えることができる。ユーザは最適化プロセスを指示するため目的コード内に注釈を追加または変更することができる。このプロセスは、それ以上の最適化が見いだされなくなるまで、または最適化されたアプリケーションの処理性能がユーザにとって満足できるものとなるまで、反復プロセスとして実施することができる。

【００１１】

【発明の実施の形態】図１は、動的変換ルーチン１５を使用してアプリケーション１０を実行するコンピュータ・システム７のブロック図である。コンパイラが実行時の前に実行可能コードを生成するのに対し、動的変換ルーチン１５は実行時に変換されたコード８を生成する点がコンパイラと相違する。アプリケーション１０のコード・ブロックが実行時に変換される。変換されたコード・ブロックは、図１において変換されたコード８として示されている。変換されたコード８は、何度も実行されるコード・ブロックの各々が実行時に一度だけ変換されるだけでよいように、キャッシュ・メモリのようなメモリに記憶される。このアプローチは、実行前にコードが変換されることを必要としない柔軟性を実現するだけでなく、実行毎にコード・ブロックを変換しなければならないとした場合に生じるオーバーヘッドをも減少させる。

【００１２】本発明の好ましい実施形態において、変換されたコード８は変換コード・キャッシュ２０に記憶される。変換コード・キャッシュ２０が一杯の時、新しく変換されるコード・ブロックのために空間を作成するため以前に変換されたコード・ブロックの一部が破棄される必要があるかもしれない。これは、破棄されたコード・ブロックが再使用されるならば再変換されるべきことを必要とするが、メモリ使用における潜在的節約を可能にする。以前に変換されたコード・ブロックを破棄する代わりに、それらをシステム・メモリに記憶することもできる。

【００１３】図２は、本発明の好ましい実施形態に従ってプロファイル・データを生成する動的変換ルーチンを利用するシステムの流れ図である。ソース・コード３１は、ステップ３２において、アプリケーション３３を作成するためコンパイルされる。コンパイルされたアプリケーション３３は、ステップ３４で、テストされ妥当性を検査されることもある。アプリケーション３３は、テストおよび妥当性検査の間全速力で実行できる。コンパイル時にアプリケーション３３にプロファイル作成コードが加えられなかったので、アプリケーション３３の処理性能は、アプリケーション３３内へのプロファイル作成用コードの内包によって妨げらることがない。

【００１４】ステップ３５において、プロファイル情報を収集するためアプリケーション３３は動的変換ルーチンの下で実行される。変換ルーチンは目的コードから目的コードへの変換を実行する。すなわち、アプリケーション３３の目的コードは、変換ルーチンによって変換された目的コードに変換される。次に変換された目的コードが実行される。ステップ３５で生成される変換された目的コードとステップ３２でアプリケーション３３を作成するためコンパイルされた目的コードの間の相違は、プロファイル作成コードがステップ３５で作成された変換目的コードに加えられる点である。換言すれば、ステップ３５で生成された変換目的コードが、実行時にプロファイル情報３６を生成するために使用されるプロファイル作成コードを付加的に含む点を除いて、ステップ３５で生成された変換目的コードはステップ３２でアプリケーション３３を作成するためコンパイルされた目的コードと同等または本質的に同じである。

【００１５】ステップ３７において、プロファイル情報３６を使用して第２のコンパイルの間最適アプリケーション３８が作成される。要するに、プロファイルに基づく最適化（ＰＢＯ）処理の実行のためアプリケーション・

ソース・コードは２回だけコンパイルされる。第１回は、アプリケーションをテストし妥当性を検査するために使用されるアプリケーションのソース・コード・コピーを作成するためである。アプリケーションをテストし妥当性を検査するために使用されたものと同じアプリケーション・コピーがプロファイル・データを生成するため変換ルーチンの下で実行される。次に、コンパイルされたアプリケーションを更に最適化するためプロファイル・データを使用してアプリケーションを構築する第２のコンパイルが実行される。

【００１６】図３は、アプリケーション１０のコードを動的に変換して実行する動的変換および実行プロセス３９の使用形態を示している。変換は、プロファイル作成コードを備えた変換目的コードを生成する。アプリケーション１０は、命令１１およびデータ１２を含むように示されている。アプリケーション１０内の命令１１が実行されるにつれて、ステップ１６において、プログラム・カウンタがアプリケーション１０内でコード１３の新しいブロックをポイントする度毎に、その新たなコード・ブロックが動的変換および実行プロセス３９によって既に変換されたものであるか否かが判断される。変換済みでなければ、ステップ１８において、そのコード・ブロックはホスト・コンピュータ・システムに対して適切な目的コードに変換される。加えて、プロファイル・データを生成するプロファイル作成コードが追加される。上述のように、変換されるべきコード・ブロックは既にホスト・コンピュータ・システム固有のものとなっているので、変換は、プロファイル作成コードを目的コードに組み入れるためだけに行われる目的コードから目的コードへの変換にすぎない。ステップ１７において、コード・ブロック１３に関する変換されたコードが実行される。

【００１７】図４は、動的変換および実行プロセス３９によってアプリケーションを実行するプロセスを更に示している。ステップ２１でアプリケーションの実行は始まる。ステップ２２において実行されるべき次のアプリケーション・アドレスが調べられる。これは、例えばブロック情報マップへのアドレス２５を使用して行われる。ブロック情報マップへのアドレス２５はアプリケーションにおけるコード・ブロックのエントリを含み、更に、アプリケーションによって呼び出される共有ライブラリにおけるコード・ブロックのエントリを含む場合もある。本発明の好ましい実施形態においては、実行のため実際に必要とされるコード・ブロックに関するアドレスだけがブロック情報マップ２５内に含まれる。これらのアドレスは、コード・ブロックが必要とされる都度、実行時に追加される。

【００１８】コード・ブロックは、１つまたは複数の命令からなるグループである。コード・ブロックは、(「基本ブロック」と同様に)分岐によって終了する単一または直線的命令シーケンスか、プロシージャか、あるいはその他の命令群である。本発明の好ましい実施形態では、コード・ブロックは、動的変換ルーチン１５によって活用される変換の単位である。この変換の単位は、例えば、プロシージャ、基本ブロック、ページ、キャッシュ線、単一命令、またはその他の命令の組み合わせであることができる。

【００１９】ブロック情報マップへのアドレス２５の各エントリは、アプリケーション内のコード・ブロックまたは共有ライブラリ内のコード・ブロックの開始アドレスを識別する少くとも１つの第１のプログラム・アドレスを含む。加えて、各エントリは、アプリケーションまたは共有ライブラリからのコード・ブロックに関するブロック情報を含むブロック情報テーブル２６における位置をポイントするブロック情報ポインタを含む。ブロック情報テーブル２６内のブロック情報は、そのコード・ブロックに関する変換されたブロック・コードが存在する場合変換されたコードに対するポインタを含む。

【００２０】ステップ２３において、実行されるべき次のアプリケーション・アドレスを調べた後、アプリケーションまたは共有ライブラリからのコード・ブロックに関する変換されたコードが存在するか否かの判断が行われる。これは、例えば、ブロック情報テーブル２６から取り出された情報を使用して行われる。そのコード・ブロックに関して変換が存在しなければ、ステップ２４においてコード・ブロックが変換される。変換されたコードが次に実行される。

【００２１】変換コード・キャッシュ２０を使用して、効率的な実行のため変換されたコード・ブロックが保持される。変換コード・キャッシュ２０によって保持されるコード・ブロックの数は、例えば使用可能メモリに応じて変わる。図４において、変換コード・キャッシュ２０内のコード・ブロックは、コード・ブロック２７、コード・ブロック２８およびコード・ブロック２９によって表されている。これらのコード・ブロックのいずれも共有ライブラリからの変換されたプロシージャを表す場合もある。

【００２２】変換コード・キャッシュ２０内のコード・ブロックは、変換コード・キャッシュ２０内の他のコード・ブロックに分岐することもある。例えば、コード・ブロック２７内の分岐命令がコード・ブロック２８内の位置をポイントする。同様に、コード・ブロック２８のコードを実行した後、プログラムの流れ制御がコード・ブロック２９へ続く。図４で示されるように、コード・ブロック２７のコードの実行後、プログラムの流れ制御は、変換コード・キャッシュ２０の外側のコード・ブロックに続くこともある。同様に、コード・ブロック２９のコードの実行後、動的分岐が、変換コード・キャッシュ２０の外側のコード・ブロック内のアドレスへ分岐する。

【００２３】変換コード・キャッシュ２０の範囲内のコード・ブロックの命令の実行が一旦開始すると、変換コード・キャッシュ２０の範囲内のそのコード・ブロックおよび他のコード・ブロックの命令の実行は、実行すべき命令が変換コード・キャッシュ２０の範囲外のものとなるまで、継続する。キャッシュ・ミスが発生すると、プログラム制御は、実行されるべき次のアプリケーション・アドレスが調べられるステップ２２に戻る。(変換済みでなければ)該当するコード・ブロックが変換され、変換コード・キャッシュ２０に置かれ、アプリケーションの実行は続く。

【００２４】図５は、本発明の１つの代替実施形態に従ってプロファイル・データを生成する動的変換を活用するシステムの動作の流れを示している。ステップ４２において、ソース・コード４１が最適化されたアプリケーション４３を作成するようにコンパイルされる。コンパイルの間、最適化されたアプリケーションの実行時動作を照会する機能を持つコード注釈が追加される。プロファイル情報が使用可能な場合、それはアプリケーションを最適化するために使用される。プロファイル情報は、また、一層詳細な実行時動作を照会する新しいコード注釈の追加を促すために使用される。

【００２５】ステップ４５において、変換ルーチンの下でアプリケーション４３が実行され、プロファイル情報が収集される。変換ルーチンは、目的コードから目的コードへの変換を実行する。すなわち、アプリケーション４３の目的コードは、変換ルーチンによって、変換目的コードに変換される。次に変換目的コードが実行される。ステップ４４で生成された変換目的コードは、ステップ４２でアプリケーション４３を作成するためコンパイルされた目的コードと同じ機械命令セットを使用する。テップ４４で生成された変換目的コードとステップ４２でアプリケーション４３を作成するためコンパイルされた目的コードの相違は、コンパイルされたアプリケーションにおける注釈が、実行時にプロファイル情報４５を生成するプロファイル作成コードと置き換えられる点である。追加されるプロファイル作成コードは、最適化されたアプリケーション４３内の注釈に基づいて追加される。

【００２６】プロセスは、最適化されたアプリケーション４３を更に最適化するように繰り返すこともできる。最適化されたアプリケーション４３の各動的変換および実行から得られるプロファイル情報４５が、次のコンパイルでアプリケーション４３を更に最適化するために使用される。注釈は、将来のコンパイルで使用される付加的プロファイル情報を得るように各コンパイル毎に調節される。最適化されたアプリケーション４３に対する最適化がそれ以上行われなくなるまで、あるいは、最適化がアプリケーション４３の処理性能がその最適化アプリケーションの開発者／テスターを満足させるまで、このような反復プロセスは続く。

【００２７】図６は、本発明の別の好ましい代替実施形態に従って動的変換ルーチンを利用してプロファイル・データを生成するシステムの動作の流れを示す。ステップ５２でソース・コード５１が最適化されたアプリケーション５３を作成するためコンパイルされる。コンパイルの間、最適化されたアプリケーションの実行時動作を照会する機能を持つコード注釈が追加される。プロファイル情報が使用可能な場合アプリケーションを最適化するために使用される。更に、プロファイル情報は、一層詳細な実行時動作を照会する新しいコード注釈の追加を促すために使用されることもできる。

【００２８】ステップ５６において、ユーザが注釈およびプロファイル情報の両者を調べる。そこでユーザは新しい注釈を追加することができる。ユーザのこのような操作は、コンパイルの前、その間またはその後のいずれの時点でも実行できる。ステップ５５において、プロファイル情報を収集するために、アプリケーション５３は変換ルーチンの下で実行される。変換ルーチンは、目的コードから目的コードへの変換を実行する。すなわち、アプリケーション５３の目的コードは、変換ルーチンによって変換目的コードに変換される。次に変換目的コードが実行される。ステップ５４で生成された変換目的コードは、ステップ５２でアプリケーション４３を作成するためコンパイルされた目的コードと同じ機械命令セットを使用する。テップ５４で生成された変換目的コードとステップ５２でアプリケーション５３を作成するためコンパイルされた目的コードの相違は、コンパイルされたアプリケーションにおける注釈が、実行時にプロファイル情報５５を生成するプロファイル作成コードと置き換えられる点である。追加されるプロファイル作成コードは、最適化されたアプリケーション５３内の注釈に基づいて追加される。

【００２９】プロセスは、最適化されたアプリケーション５３を更に最適化するように繰り返すこともできる。最適化されたアプリケーション５３の各動的変換および実行から得られるプロファイル情報５５が、次のコンパイルでアプリケーション５３を更に最適化するために使用される。注釈は、将来のコンパイルで使用される付加的プロファイル情報を得るように各コンパイル毎に調節される。最適化されたアプリケーション５３に対する最適化がそれ以上行われなくなるまで、あるいは、最適化がアプリケーション５３の処理性能がその最適化アプリケーションの開発者／テスターを満足させるまで、このような反復プロセスは続く。

【００３０】以下の諸表は、本発明の好ましい実施形態の動作を更に示すサンプル・コードを提供する。表１は、アプリケーションにおける１つのサンプル・プロシージャに関するソース・コードを示す。

【００３１】

【表１】ソース

行番号	コード
5	sum=0
6	for (i=0; i<array_size; i++)
7	sum += array[i]; *

```
;行５の目的コード
STW %gr0,sum(DP)            ;sumに0をストア
;行６の目的コード
STW %gr0,i(DP)              ;iを0に初期化
LDW i(DP),%tr1              ;一時レジスタにiをロード
loop_back
LDW array_size(DP),%tr2     ;一時レジスタにarray_sizeをロード
COMB,>=,n %tr2,%tr1,exit_loop  ;if(i>=array_size) ループを出る
;行７の目的コード
ADDIL array(DP),%tr3        ;arrayのアドレスを入手
LDWX,s,m %tr1,(%tr3),%tr4   ;arry[i]を入手
LDW sum(DP),%tr5            ;sumを入手
ADD %tr4,%tr5,%tr6          ;sumとarray[i]を加算
STW %tr6,0(%tr3)            ;新しいsumをストア
LDO 1(%tr1),%tr1            ;i++
B,n  loop_back              ;次の反復を実行
exit_loop
```

【００３３】表２の目的コードがプロファイル作成機能を持つ動的変換の下で実行される時、動的変換ルーチン１５は、頻繁に実行されるコード部分を識別するためプロファイル作成コードを追加する。これによってコンパイラはより高品質のコードを生成することが可能となる。これは、例えば、カウンタを分岐に関連づけること(弓形カウント)、あるいはカウンタを各行を表すコード※に関連づけること(行頻度カウント)のいずれかによって実施される。

＊次の表２は、表１のソース・コードがコンパイルされた結果の目的コードを例示する。

【００３２】

【表２】

```
;行５の目的コード
STW %gr0,sum(DP)            ;sumに0をストア
;行６の目的コード
STW %gr0,i(DP)              ;iを0に初期化
LDW i(DP),%tr1              ;一時レジスタにiをロード
loop_back
LDW array_size(DP),%tr2     ;一時レジスタにarray_sizeをロード
COMB,>=,n %tr2,%tr1,exit_loop  ;if(i>=array_size) ループを出る
;行７の目的コード
ADDIL array(DP),%tr3        ;arrayのアドレスを入手
LDWX,s,m %tr1,(%tr3),%tr4   ;arry[i]を入手
LDW sum(DP),%tr5            ;sumを入手
ADD %tr4,%tr5,%tr6          ;sumとarray[i]を加算
STW %tr6,0(%tr3)            ;新しいsumをストア
LDO 1(%tr1),%tr1            ;i++
B,n  loop_back              ;次の反復を実行
exit_loop
```

【００３４】次の表３は、弓形カウントを実行するように動的変換ルーチン１５によって追加されたプロファイル作成コードを組み込まれた表２の目的コードを示す。

【００３５】

【表３】

```
;行５の目的コード
STW %gr0,sum(DP)            ;sumに0をストア
;行６の目的コード
;プロファイル作成コード(パート１)
LDW Line6Hit(DP),%tr8       ;行６を通過した回数を示すカウンタを入手
ADDI,<>1,%tr8,%tr8          ;カウンタを増分
LDI -1,%tr8                 ;オーバーフローの場合カウンタを一杯にする
STW %tr8,Line6Hit(DP)       ;カウンタをストア
STW %gr0,i(DP)              ;iを0に初期化
LDW i(DP),%tr1              ;一時レジスタにiをロード
loop_back
LDW array_size(DP),%tr2     ;一時レジスタにarray_sizeをロード
COMB,>=,n %tr2,%tr1,exit_loop  ;if(i>=array_size) ループを出る
;プロファイル作成コード(パート２)
LDW Line6NotTaken(DP),%tr8      ;行６を通過した回数を示すカウンタを入手
ADDI,<>1,%tr8,%tr8          ;カウンタを増分
LDI -1,%tr8                 ;オーバーフローの場合カウンタを一杯にする
STW %tr8,Line6NotTaken(DP)      ;カウンタをストア
;行７の目的コード
ADDIL array(DP),%tr3        ;arrayのアドレスを入手
LDWX,s,m %tr1,(%tr3),%tr4   ;arry[i]を入手
```

```
LDW sum(DP),%tr5          ;sumを入手
ADD %tr4,%tr5,%tr6        ;sumとarray[i]を加算
STW %tr6,0(%tr3)          ;新しいsumをストア
LDO 1(%tr1),%tr1          ;i++
B,n  loop_back            ;次の反復を実行
exit_loop
```

【００３６】表３の目的コードが実行される時、コードは、オリジナルのソースの中の各ステートメントが実行された回数の追跡を続ける。この情報がコンパイラにフィードバックされると、コンパイラは、場合によっては、行６および７のループに関する一層詳細な情報を使用すべきか否か判断する。その場合には、コンパイラはコードに注釈を付加する。この注釈は実際には目的コード＊に現れないが、(シンボル・テーブルまたはデバッギング情報のように)非実行部分のアプリケーション・ファイルに含めることができる。例えば、次の表４に示されるように、コンパイラは、行６に関する詳細なループ情報を要求する注釈をファイルに記述する。

【００３７】

【表４】

位置	アクション	パラメータ
<Label_X>:	Loop_Detai:	Iteration_Bound=(array_size-1)

【００３８】表４に記述された注釈は、位置、アクションおよびアクションを実行するために必要とされるオプションとしての付加パラメータという３つの部分から成る。この例における位置は、コンパイラによって生成された目的コードにおける位置を表すラベルである。アクションは、実行中に上記位置に出会う度毎に情報を収集するため動的変換ルーチン１５によって解読されるべきコードである。この例では、アクションは、例えば反復動作の２、４または８による分割可能性やループの「モード」等Label_ Xが付けられたループに関する詳細を収集することである。このアクションを実行するため、動※的変換ルーチン１５は、反復の境界がどこに位置しているか知る必要がある。上述の例では、反復の境界は、変数"array_size"(アレイの大きさ)に含まれていて、１を減ずることによって調節する必要がある。

【００３９】次の表５は、注釈を使用する変換されたコードおよびプロファイル作成コードのサンプルである。このコードが動的変換の下で実行される時、動的変換ルーチン１５は、Label_Xの位置にループ詳細分析を行うコードを挿入する。

【００４０】

【表５】

```
;行５の目的コード
STW %gr0,sum(DP)          ;sumに0をストア
;行６の目的コード
;注釈[<Label_X><Loop_Detail><array_size-1>]に関する
;プロファイル作成コード
LDW array_size(DP),%arg0  ;ループ反復限界を入手
BL Analyze_Loop,RP        ;分析プロシージャを呼び出す
ADDI -1,%arg0,%arg0       ;反復限界を調整
STW %gr0,i(DP)            ;iを0に初期化
LDW i(DP),%tr1            ;一時レジスタにiをロード
loop_back
LDW array_size(DP),%tr2   ;一時レジスタにarray_sizeをロード
COMB,>=,n %tr2,%tr1,exit_loop  ;if(i>=array_size) ループを出る
;行７の目的コード
ADDIL array(DP),%tr3      ;arrayのアドレスを入手
LDWX,s,m %tr1,(%tr3),%tr4 ;arry[i]を入手
LDW sum(DP),%tr5          ;sumを入手
ADD %tr4,%tr5,%tr6        ;sumとarray[i]を加算
STW %tr6,0(%tr3)          ;新しいsumをストア
LDO 1(%tr1),%tr1          ;i++
B,n  loop_back            ;次の反復を実行
exit_loop
```

【００４１】表３および表５のサンプル・コードによって、目的コードが実行されるとプロファイル情報が生成される。この情報はコンパイラもユーザも利用することができる。ユーザは"Loop-Detail"のような新しい注釈

を加えることによって、追加情報を要求するか、コンパイラが所望のコードを生成するようにコンパイラにヒントを与えることことができる。

【００４２】以上は本発明の典型的な実施形態を開示し記述したものにすぎない。本発明をその理念および基本的特徴から逸脱することなく別の形態で実施することが可能である点は当業者に認められるべきものであろう。

【００４３】本発明には、例として次のような実施様態が含まれる。

（１）アプリケーションに関する最適化された目的コードを生成する方法であって、アプリケーションのソース・コードをコンパイルして該アプリケーションに関する第１の目的コードを生成するステップ(a)と、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを含む第２の目的コードに上記第１の目的コードを変換し(b1)、該第２の目的コードを実行してプロファイル情報を生成する(b2)ことを含む上記第１の目的コードに関するプロファイル情報を生成するステップ(b)と、上記プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成するステップ(c)と、を含む最適目的コード生成方法。

（２）上記ステップ(a)に続いて、該アプリケーションに関して上記第１の目的コードをテストし、妥当性を検査するステップ(d)を更に含む上記（１）に記載の最適目的コード生成方法。

（３）上記ステップ(c)に続いて、実行の際に追加プロファイル情報を生成する機能を持つプロファイル作成コードを含む第３の目的コードに上記最適化された目的コードを変換し(d1)、該第３の目的コードを実行して追加プロファイル情報を生成する(d2)ことを含む上記最適化された目的コードに関する追加プロファイル情報を生成するステップ(d)と、上記追加プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、更に最適化された目的コードを生成するステップ(e)と、を含む上記（１）に記載の最適目的コード生成方法。

（４）上記ステップ(a)が上記第１の目的コードに注釈を挿入することを含む、上記（１）に記載の最適目的コード生成方法。

（５）上記ステップ(b1)において、第１の目的コード内の注釈が第２の目的コード内のプロファイル作成コードの内容を決定する、上記（４）に記載の最適目的コード生成方法。

（６）上記ステップ(a)において、ソース・コードをコンパイルするコンパイラによって注釈が第１の目的コードに置かれる、上記（５）に記載の最適目的コード生成方法。

（７）上記ステップ(a)において、ユーザによって注釈が第１の目的コードに置かれる、上記（５）に記載の最適目的コード生成方法。

（８）上記ステップ(c)に続いて、実行の際に追加プロファイル情報を生成する機能を持つプロファイル作成コードを含む第３の目的コードに上記最適化された目的コードを変換し(d1)、該第３の目的コードを実行して追加プロファイル情報を生成する(d2)ことを含む上記最適化された目的コードに関する追加プロファイル情報を生成するステップ(d)と、上記追加プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、更に最適化された目的コードを生成するステップ(e)と、を含む上記（５）に記載の最適目的コード生成方法。

（９）上記第１の目的コードおよび上記第２の目的コードが同じ機械命令セットを利用する、上記（１）に記載の最適目的コード生成方法。

【００４４】（１０）アプリケーションに関する第１の目的コードを第２の目的コードに変換し、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを該第２の目的コードに追加する変換手段と、該第２の目的コードを実行してプロファイル情報を生成する手段と、を備えるコンピュータ・システム。

（１１）該アプリケーションに関して上記第１の目的コードをテストし、妥当性を検査する手段を更に備える、上記（１０）に記載のコンピュータ・システム。

（１２）ソース・コードをコンパイルして上記第１の目的コードを生成するコンパイラを更に備える、上記（１０）に記載のコンピュータ・システム。

（１３）上記第２の目的コードの実行によって生成された上記プロファイル情報を利用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成する手段を備える、上記（１２）に記載のコンピュータ・システム。

（１４）上記コンパイラが上記第１の目的コードに注釈を挿入する、上記（１２）に記載のコンピュータ・システム。

（１５）上記変換手段が上記第１の目的コード内の上記注釈を利用して上記第２の目的コード内の上記プロファイル作成コードの内容を決定する、上記（１４）に記載のコンピュータ・システム。

（１６）ユーザが上記注釈を第１の目的コードに置く手段を含み、上記変換手段が上記第１の目的コード内の該注釈を利用して上記第２の目的コード内の上記プロファイル作成コードの内容を決定する、上記（１２）に記載のコンピュータ・システム。

（１７）上記第２の目的コードのサブセットを持つモジュールを記憶する変換コード・キャッシュを更に備える上記（１０）に記載のコンピュータ・システム。

（１８）ブロック情報マップへのアドレスと、ブロック情報テーブルと、を更に備える上記（１７）に記載のコンピュータ・システム。

15

（１９）上記第１の目的コードおよび上記第２の目的コードが同じ機械命令セットを利用する、上記（１０）に記載のコンピュータ・システム。

【００４５】

【発明の効果】本発明によって、プロファイル作成コードを備えた特別版の目的コードを生成するため独立したコンパイルを実行する必要性なしに、プロファイルに基づくプログラムの最適化が可能になる。プロファイル・データを生成するため、本発明に従った動的変換ルーチンを使用することによって、アプリケーション開発者のＰＢＯ使用が単純化される。更に、本発明の動的変換システムは、動的データ収集に容易に対応でき、ＰＢＯに対する反復的アプローチにおける注釈の使用を可能にする。これによって、一層効率的で大規模な実行時プロファイル・データ収集が可能になる。

【図面の簡単な説明】

【図１】本発明の好ましい実施形態に従う動的変換ルーチンを含むコンピュータ・システムのブロック図である。

【図２】本発明の好ましい実施形態に従って動的変換ルーチンを利用してプロファイル・データを生成するシステムの動作の流れ図である。

【図３】本発明の好ましい実施形態に従ってコードを動的に変換する動的変換ルーチンの使用を示すブロック図である。

16

【図４】本発明の好ましい実施形態に従う動的変換ルーチンによって使用されるブロック情報マップへのアドレス、ブロック情報テーブルおよび変換されたコード・キャッシュを示すブロック図である。

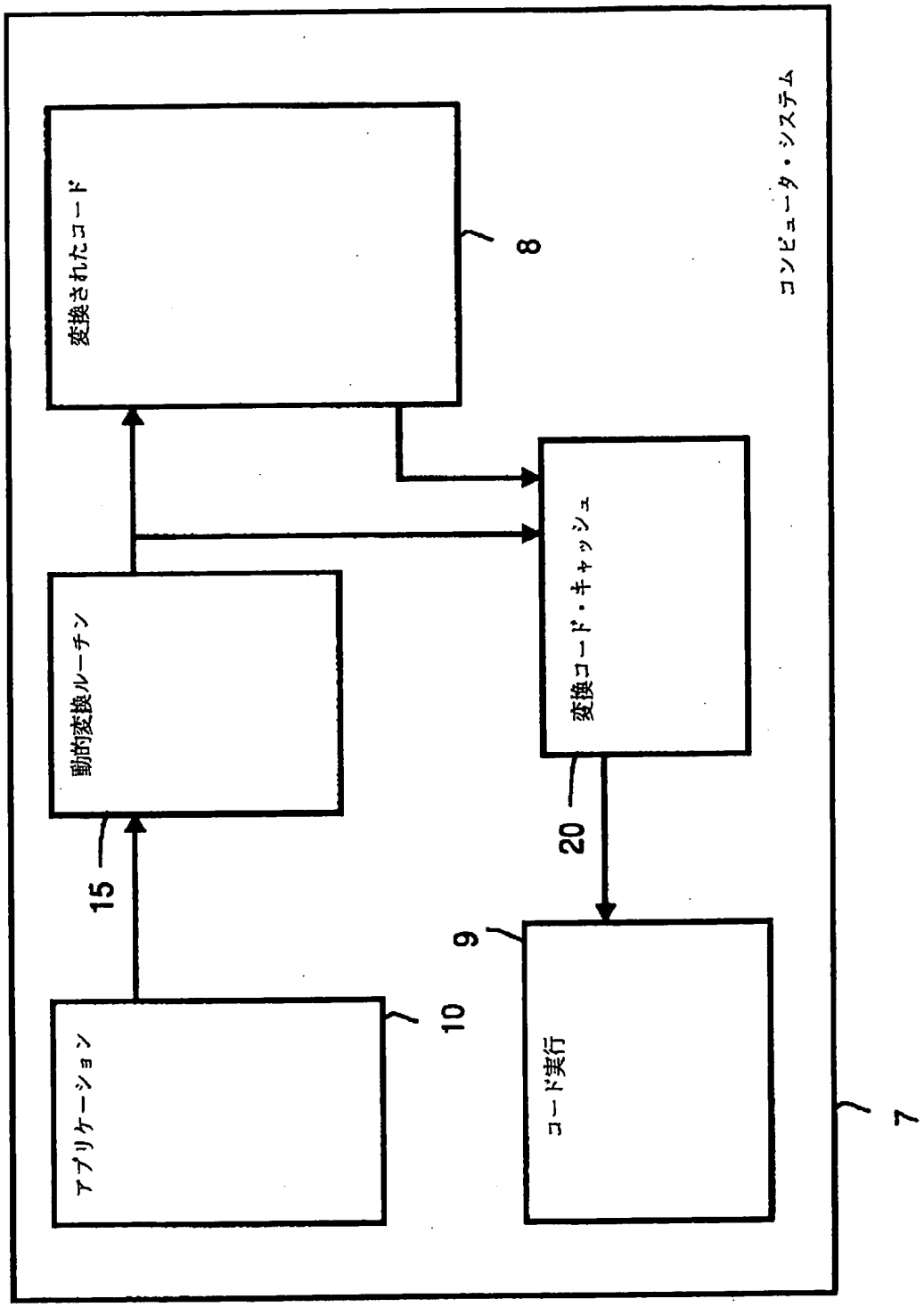
【図５】本発明の好ましい実施形態に従って動的変換ルーチンおよびコード注釈を利用してプロファイル・データを生成するシステムの動作の流れ図である。

【図６】本発明の好ましい実施形態に従って動的変換ルーチンおよびユーザ修正可能コード注釈を利用してプロファイル・データを生成するシステムの動作の流れ図である。

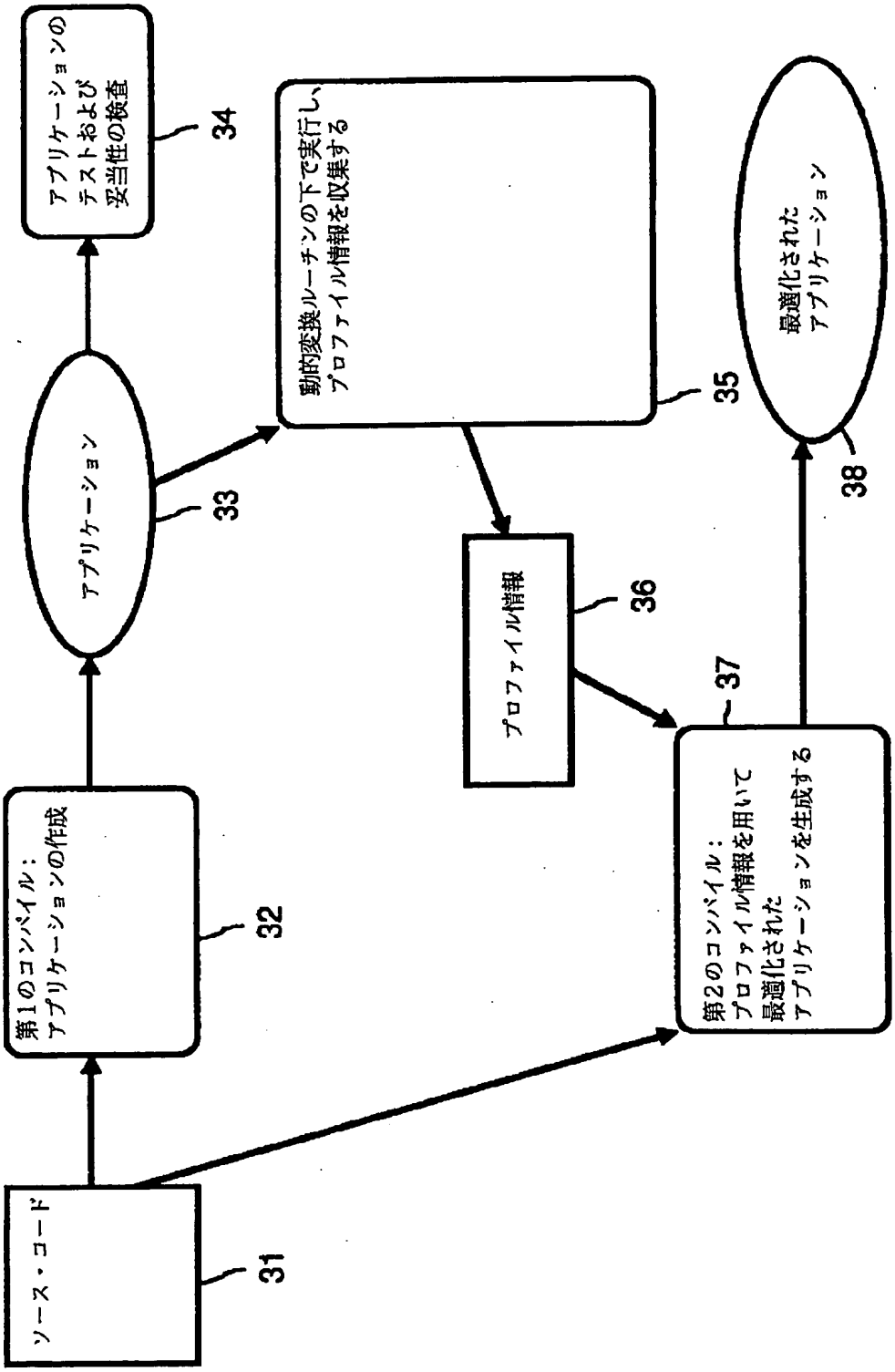
【符号の説明】

７　　コンピュータ・システム
１０、３３　　アプリケーション
１５、４４、５４　　動的変換ルーチン
２０　　変換コード・キャッシュ
２５　　ブロック情報マップへのアドレス
２６　　ブロック情報テーブル
２７、２８、２９　　変換目的コード
３１、４１、５１　　ソース・コード
３６、４５、５５　　プロファイル情報
３８、４３、５３　　最適化されたアプリケーション
３９　　動的変換・実行プロセス
４２、５２　　コンパイラ
５６　　ユーザ

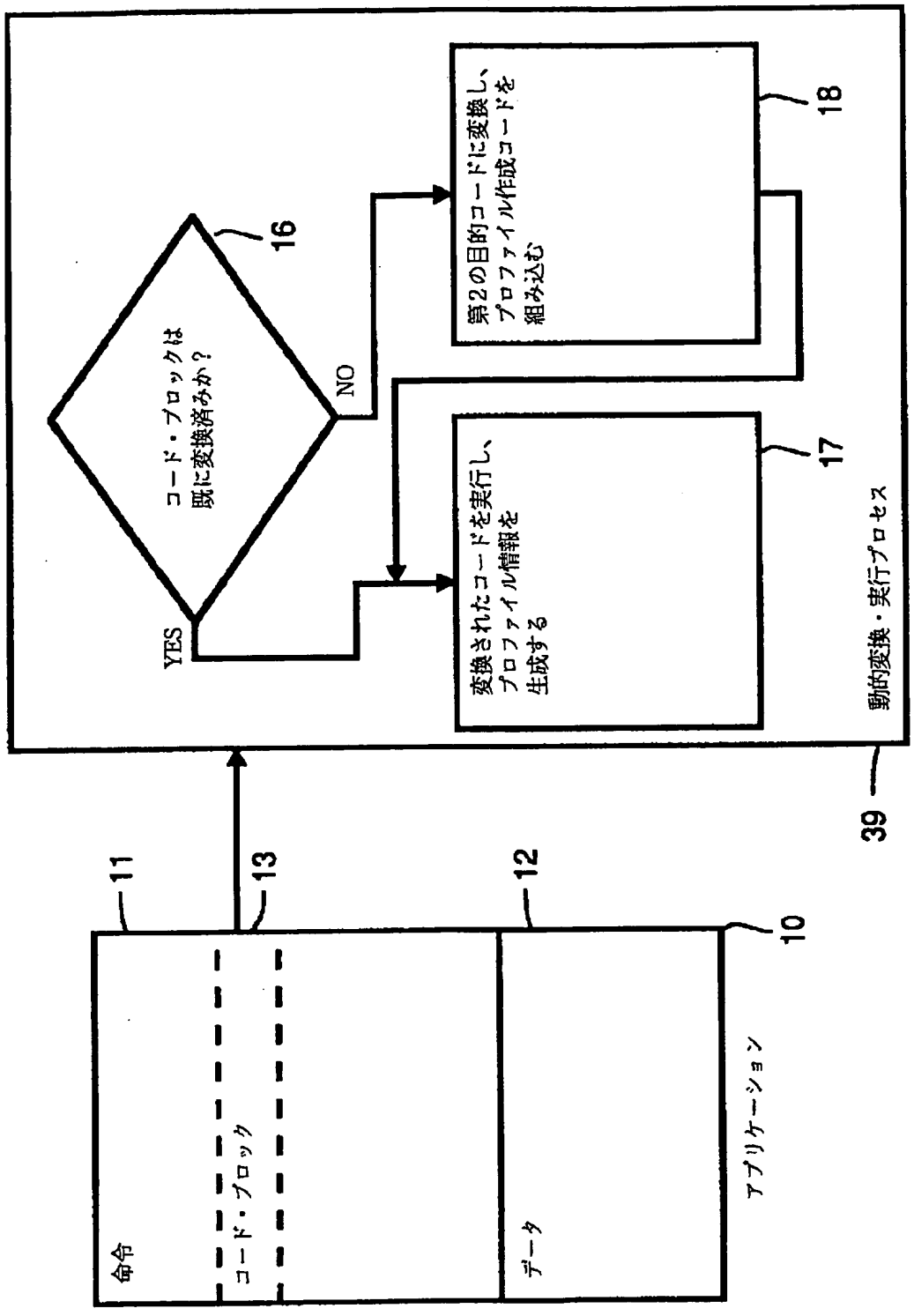
【図１】


コンピュータ・システム
変換されたコード
8
動的変換ルーチン
変換コード・キャッシュ
15
20
9
10
アプリケーション
コード実行
7

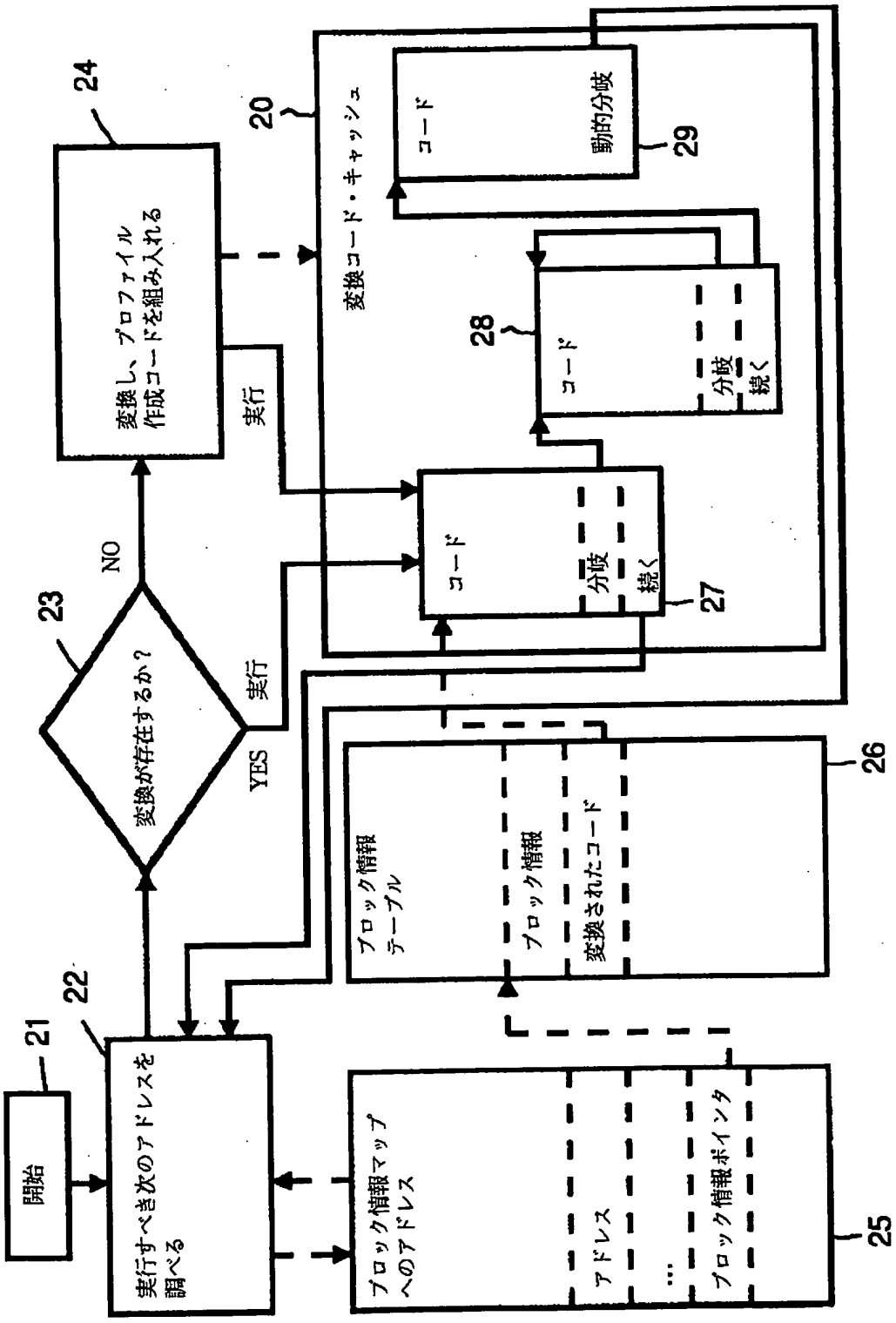
【図２】


ソース・コード
31
第1のコンパイル：
アプリケーションの作成
32
アプリケーション
33
アプリケーションの
テストおよび
妥当性の検査
34
動的変換ルーチンの下で実行し、
プロファイル情報を収集する
35
プロファイル情報
36
第2のコンパイル：
プロファイル情報を用いて
最適化された
アプリケーションを生成する
37
最適化された
アプリケーション
38

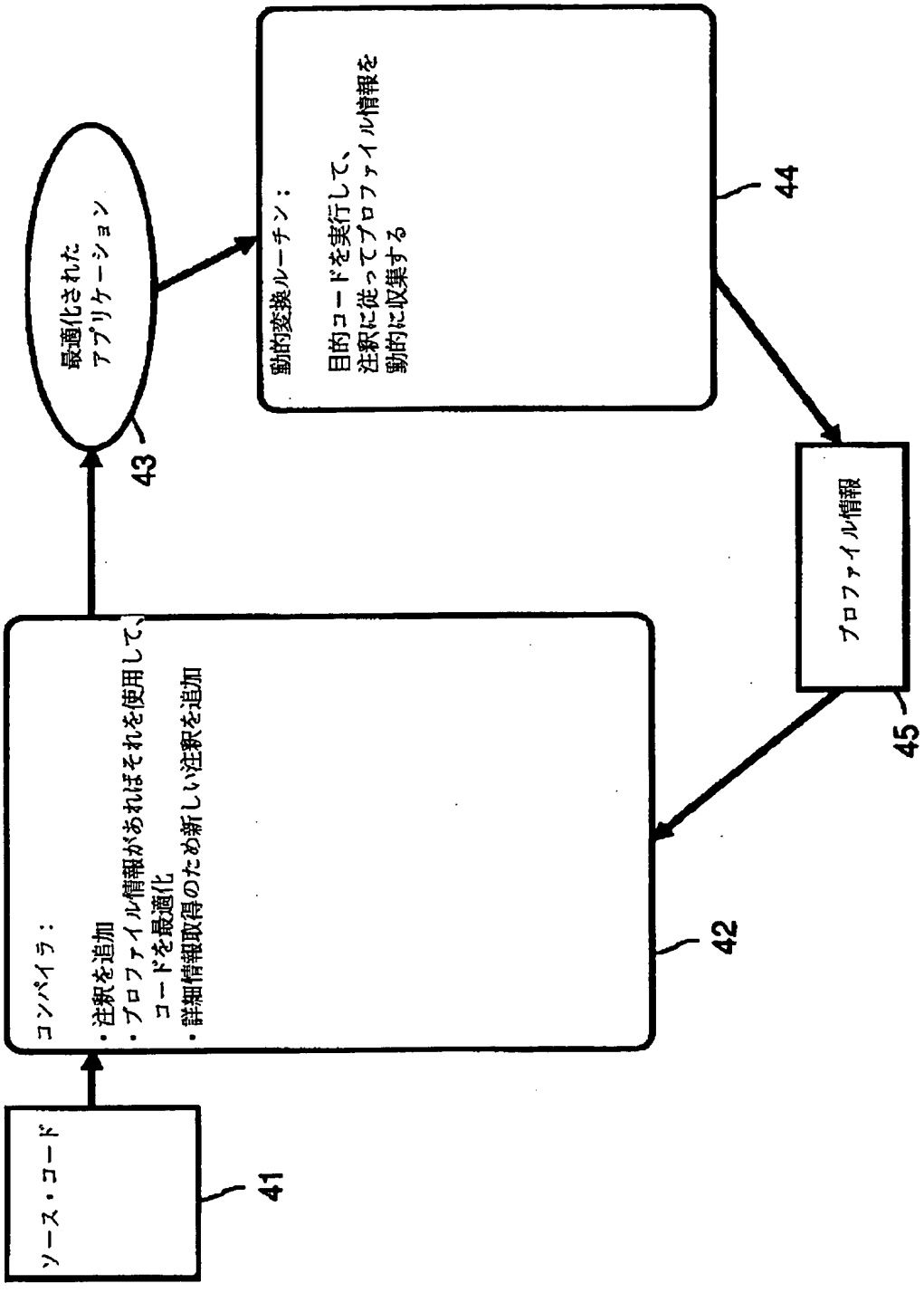
【図３】


命令
コード・ブロック
データ
アプリケーション
10
11
12
13
39
動的変換・実行プロセス
コード・ブロックは
既に変換済みか？
16
YES
NO
変換されたコードを実行し、
プロファイル情報を
生成する
17
第2の目的コードに変換し、
プロファイル作成コードを
組み込む
18

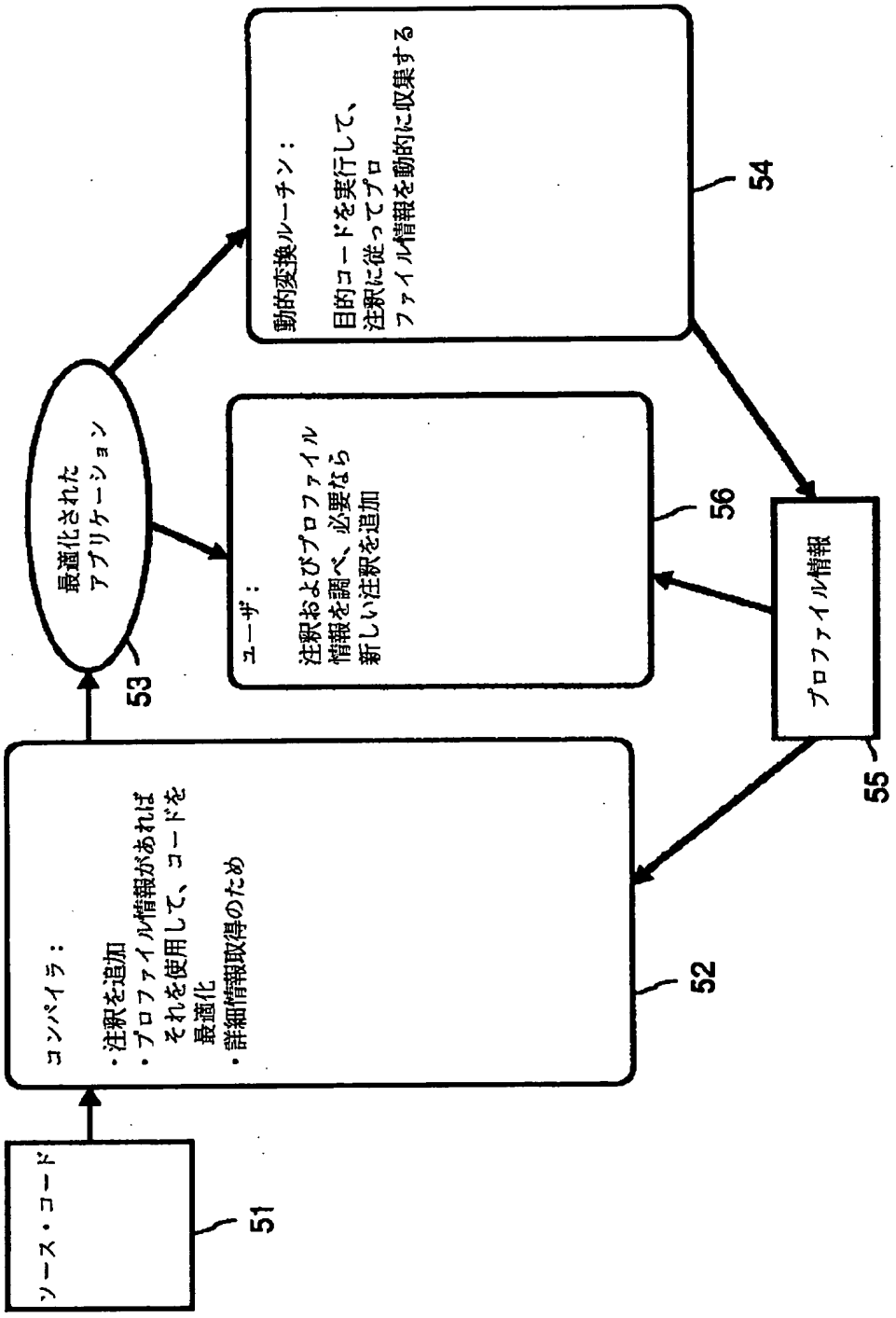
【図４】


開始
21
実行すべき次のアドレスを調べる
22
変換が存在するか？
23
NO
YES
変換し、プロファイル作成コードを組み入れる
24
実行
実行
変換コード・キャッシュ
20
コード
動的分岐
29
コード
分岐
続く
28
コード
分岐
続く
27
ブロック情報テーブル
ブロック情報
変換されたコード
26
ブロック情報マップへのアドレス
アドレス
...
ブロック情報ポインタ
25

【図５】


ソース・コード
41
コンパイラ：
・注釈を追加
・プロファイル情報があればそれを使用して、コードを最適化
・詳細情報取得のため新しい注釈を追加
42
最適化された
アプリケーション
43
動的変換ルーチン：
目的コードを実行して、
注釈に従ってプロファイル情報を
動的に収集する
44
プロファイル情報
45

【図６】


ソース・コード
51
コンパイラ：
・注釈を追加
・プロファイル情報があれば
それを使用して、コードを
最適化
・詳細情報取得のため
52
最適化された
アプリケーション
53
動的変換ルーチン：
目的コードを実行して、
注釈に従ってプロ
ファイル情報を動的に収集する
54
ユーザ：
注釈およびプロファイル
情報を調べ、必要なら
新しい注釈を追加
56
プロファイル情報
55